A variant of the Recoil Growth algorithm to generate multi-polymer systems

Florian Simatos

A VARIANT OF THE RECOIL GROWTH ALGORITHM TO GENERATE MULTI-POLYMER SYSTEMS

FLORIAN SIMATOS

Abstract. The Recoil Growth algorithm, proposed in 1999 by Consta *et al.*, is one of the most efficient algorithm available in the literature to sample from a multi-polymer system. Such problems are closely related to the generation of self-avoiding paths. In this paper, we study a variant of the original Recoil Growth algorithm, where we constrain the generation of a new polymer to take place on a specific class of graphs. This makes it possible to make a fine trade-off between computational cost and success rate. We moreover give a simple proof for a lower bound on the irreducibility of this new algorithm, which applies to the original algorithm as well.

Contents

1. Introduction

Designing an algorithm that efficiently samples from a multi-polymer system according to a given probability distribution is the focus of much research activity in chemical physics [2, 10]. The state of a multi-polymer system being a collection of self-avoiding paths that do not overlap each other, this problem is closely related to the classical problem in computer science of generating self-avoiding paths. The state space of such systems is huge, especially in high dimension or if the paths are long, which makes these problems hard. The main issue consists of defining an algorithm that both keeps the computational cost low and converges rapidly to the sampling distribution. For multi-polymer systems, various approaches have been suggested to tackle this problem, among which is the Recoil Growth (RG) algorithm, meant to be one of the most efficient algorithm currently available in the literature. For a precise description of this algorithm, the reader is referred to [3][1]. In the present paper (which is an extended version of [12]), we define and analyze a variant of the RG algorithm, to which we refer as the RG* algorithm. The main ideas of both the RG and the RG* algorithms are to be found in two important classes of algorithms, which we briefly describe below: the Metropolis algorithm [9] and the auxiliary variable method [1, 7].

Date: July 2, 2009.

[1]The present paper is self-contained, and does not require prior knowledge on the RG algorithm.

The Metropolis algorithm. The Metropolis algorithm is a very generic algorithm that approximately samples according to a probability distribution π defined on some finite state space $\mathcal{X}$. It takes as other input a Markov Chain P, and constructs a reversible Markov Chain with π as stationary distribution. In the long-run, we can therefore sample from a distribution that is arbitrarily close to π. The implicit idea is that it is hard to sample from π, whereas it is easy to generate P, for instance by local modifications of a state. The new Markov Chain is built thanks to the following rejection procedure, in which we use the notation $A(x,y) = \pi(x)P(x,y)$. Starting from $x \in \mathcal{X}$, choose as candidate for the next state some $y \in \mathcal{X}$ with probability $P(x,y)$. If $A(y,x) \geq A(x,y)$, then go to y. Otherwise, flip a coin with bias $A(y,x)/A(x,y)$: if tails, go to y, and if heads, stay in x. By doing so, the probability of going from x to $y \neq x$ is $P(x,y) \cdot \min\big(1, A(y,x)/A(x,y)\big)$, and it is easy to see that this Markov chain has π as reversible distribution. An important remark is that for the rejection procedure, only the ratio $A(y,x)/A(x,y)$ matters. In particular, it is sufficient to know π up to its normalizing constant, what is very useful in many concrete applications. For instance, one could wish to sample complex combinatorial objects uniformly: in this case, one does not need to know the cardinality of these objects.

Under a more global look, the Markov Chain P can be seen as an exogenous process that, at each step of the algorithm, proposes a candidate for the next step. In general, the stationary distribution of P is not π, and so the role of the rejection procedure aims at compensating for this bias: by suitably rejecting or accepting this candidate, the new Markov Chain can be shown to be reversible with π as stationary distribution. Reversibility is an important feature here, because it makes it very easy to show that π is indeed the stationary distribution. Otherwise, since the objects under consideration are usually fairly complex, proving stationarity could be very challenging.

The auxiliary variables method. The auxiliary variable method is a conceptually easy extension of the Metropolis algorithm: instead of sampling from $\mathcal{X}$ according to π, it is sometimes easier to sample from a pair $\mathcal{X} \times \mathcal{Y}$ according to some distribution $\widetilde{\pi}(\cdot,\cdot)$ which has π as marginal. In such cases, the idea is to apply the Metropolis algorithm to $\widetilde{\pi}$, and then to recover π by summation. The new variable $y \in \mathcal{Y}$ is referred to as the "auxiliary variable". As we will see, the concept of *underlying graph* in the RG* algorithm is very close to an auxiliary variable; underlying graphs are central in the RG* algorithm, and are introduced not because they make the sampling easier or more natural, but because they reduce the computational cost.

The RG* algorithm. The state space of the RG* algorithm is the set of all possible configurations of non-intersecting polymers in some dimension d. More precisely, there is an underlying d-dimensional grid, endowed with a cyclic structure on the edges, on which all the objects are considered. A polymer is then just an undirected self-avoiding path of given length, a path being a sequence of neighboring vertices on the underlying grid. The fact that polymers cannot intersect each other, and that they cannot intersect themselves, comes from a very natural physical constraint, namely that a point in space cannot be occupied by two different particles. Figure 1 shows an example of a two-dimensional multi-polymer system on a torus with 100 polymers of length 25 each.

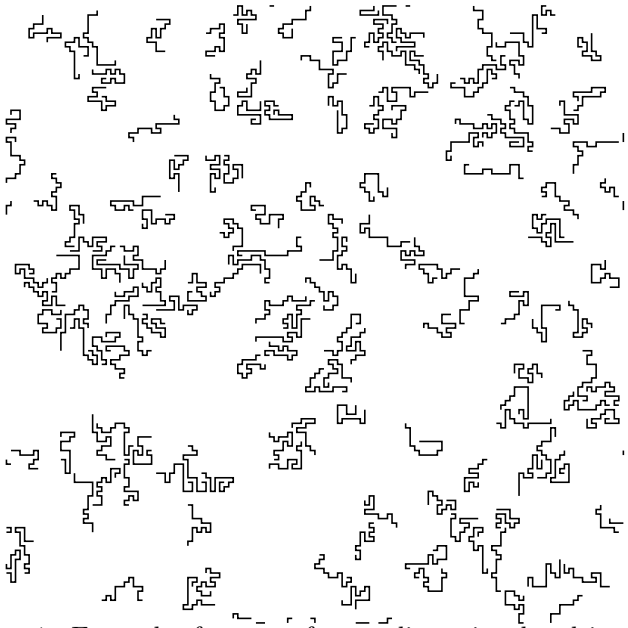


FIGURE 1. Example of a state of a two-dimensional multi-polymer system on the torus $\left(\mathbb{Z}/135\mathbb{Z}\right)^2$ with 100 polymers of length 25 each.

One step of this algorithm consists in removing one polymer at random, and trying to re-grow it elsewhere. The growth may not be successful, in which case we put back the original polymer in the system, and we remain in the same state. But if we successfully grew a candidate polymer, by analogy with the Metropolis algorithm, we flip a coin to decide whether to add this polymer or to put back the original one. The goal of this coin-tossing is to compensate for the bias introduced during the growth of the candidate polymer in order to yield the right stationary distribution. Though it does not clearly appear at this point, we want to stress the fact that computing this probability of acceptance will require a fair amount of work. This is in sharp contrast with the Metropolis algorithm, for which it is straightforward to derive this quantity.

One of the main difficulties of the RG* algorithm therefore lies in the growth of the new candidate polymer. Since polymers are self-avoiding paths, this problem is closely related to the generation of self-avoiding walks, for which an important amount of research has already been carried out [8, 13]. Two differences make the situation slightly more complex in our case. First, we have several polymers here, with the constraint that they cannot intersect each other. Second, each step of the RG* algorithm involves the attempt to grow a new polymer. In order for the algorithm to perform well, this growth needs to be done fast, even if it means that it fails often.

The efficiency of the RG* algorithm, and hence its interest compared to other algorithms, can be assessed by its ability to address a number of issues that clearly emerge from the above global description.

First, the RG* algorithm must generate a state S with probability arbitrarily close to $q(S)$, where q is the distribution we want to sample from. We have seen that this will be done by constructing a Markov chain having q as the stationary distribution. As in the Metropolis algorithm, once we have a candidate polymer, we are free to choose the probability of accepting it; we prove in this paper that the chain will be reversible for a suitable choice of the probability of acceptance. By doing so, we find that the probability of acceptance suggested in [3] is the right one. However, the same probability of acceptance fails to yield the right stationary distribution in some extended version of the algorithm, and this subtle mistake was not detected in the original paper.

Second, the algorithm must mix as quickly as possible, which means that stationarity has to be reached as fast as possible. Even on simple examples, determining the convergence rate of a Metropolis type algorithm is a very hard question, as is shown in [4]. Nevertheless, a rapidly mixing Markov chain requires both high chain construction and high acceptance rates. The *chain construction rate* refers to the probability that the growth of a new candidate polymer is successful, while the *acceptance rate* refers to the probability that, given a candidate polymer, it is accepted. We see from the above description of the RG* algorithm that if one of these two rates is low, the algorithm will stay for a long time in the same state before changing, so that it is very unlikely that it mixes rapidly.

These two quantities intrinsically depend on the generation of the new candidate polymer. For instance, we have seen that the rejection procedure of the Metropolis algorithm depends on the ratio $A(y,x)/A(x,y)$, with $A(x,y) = \pi(x)P(x,y)$: since the candidate y for the new state is chosen with probability $P(x,y)$, the question is to know whether $P(x,\cdot)$ gives advantage to states that are more likely for π. If so, then this latter ratio will often be above one, yielding a high acceptance rate. This is especially clear if P is chosen to be symmetric, in which case the acceptance probability is $\min(1, \pi(y)/\pi(x))$. In other words, a good algorithm favors candidate states that yield a good acceptance probability. As for the construction rate, a trade-off has to be made between the construction rate itself and the computational cost. Heuristically, the algorithm that generates a new polymer works by growing or recoiling one step at a time, until the polymer under construction has reached the desired length. In particular, this algorithm lives in the space of incomplete self-avoiding paths. This space is huge, especially if the dimension is large or if the polymer is long. The trade-off is therefore the following: if one allows the algorithm to visit all the space of incomplete self-avoiding paths, the construction rate will be the best possible, but the computational cost may be incredibly expensive. By truncating the set of incomplete self-avoiding paths which the algorithm is allowed to visit, one reduces this computational cost at the expense of reducing the construction rate as well.

The RG* algorithm has two salient features that yield both high construction and acceptance rates: the concepts of *feeler* and of *underlying graph*. Though these two concepts are combined in the RG* algorithm, it is worth emphasizing that they can be considered and defined independently from one another. The idea of feeler makes it possible to define an algorithm that generates a polymer step by step by looking a few steps ahead. This way, the algorithm is able to detect dense areas where it becomes hard to grow the polymer, and, if necessary, to recoil from such

regions. The feeler is characterized by a parameter ℓ, the length of the feeler, that tells how far away the algorithm is allowed to look. Large values of ℓ yield a better construction rate, but a higher computational cost.

As we will see, the algorithm that attempts to grow a new polymer works by increasing or decreasing a partial polymer by only one vertex at a time. In particular, this algorithm can be defined on any oriented graph: all that matters is to know the potential growth directions, or equivalently, the neighbors of the endpoint of the current partial polymer. However, the properties of the algorithm, namely the computational cost and the construction and acceptance rates, are crucially affected by the graph on which it is run.

The set of underlying graphs is the class of graphs on which the growth procedure, that makes use of the concept of feeler, will be run. Intuitively, we see that the more neighbors are allowed at each step, the higher the computational cost. Initially, everything happens on a d-dimensional grid, in which every vertex has $Q = 2d$ neighbors. An idea to lower the computational complexity is to impose vertices in an underlying graph to have the same out-degree $k < Q$, where k is a parameter of our algorithm to be adjusted.

With these remarks in mind, a simple sketch of the RG* algorithm is as follows. First, choose an old polymer and remove it from the system. Then generate at random an underlying graph, and try to grow a candidate polymer on this underlying graph using the idea of feeler. If the growth is not successful, then go back to the initial state. Otherwise, toss a coin to determine whether to accept or not this candidate. Clearly, the generation of an underlying graph conditions the further growth of the candidate polymer. In particular, the probability of acceptance depends on the underlying graph generated, which makes the connection with the auxiliary variable method. As we will see, delving into the details of every of these steps will require a lot of work. It is however very important to keep this simple sketch in mind.

The rest of the paper is organized as follows. In Section 2, we define the notations and the global framework, and give an overview of the concepts of underlying graph and feeler by comparing the RG* algorithm to two simpler algorithms. Section 3 then explains the RG* algorithm in details and computes the generating probabilities of the objects under consideration. We then prove in Section 4 that the Markov Chain run by the algorithm has the right stationary distribution, and find the correct probability of acceptance. Some considerations about the irreducibility of the algorithm are given in Section 4.2, and some possible extensions of the RG* algorithm are studied in Section 4.3. Finally, Section 5 is devoted to issues related to practical implementations of the algorithm.

2. Framework

2.1. **General settings, notations.** In all that follows, we are working on a finite d-dimensional lattice $\mathcal{G}$ in which each vertex has $Q = 2d$ neighbors. $\mathcal{G}$ will be referred to as the *underlying lattice*. To make sure that the lattice is finite and that each vertex has exactly Q neighbors, $\mathcal{G}$ is endowed with a torus structure. So if one thinks of $\mathcal{G}$ as a cube in dimension d, some vertices "go around". For $d = 1$, $\mathcal{G}$ is a circle, and for $d = 2$, it is a torus.

More formally, the set of vertices of $\mathcal{G}$ is $(\mathbb{Z}/a\mathbb{Z})^d$ for some fixed a, and two vertices are neighbors in $\mathcal{G}$ if all their coordinates are equal, except for one in which they differ by 1 or -1 (in $\mathbb{Z}/a\mathbb{Z}$).

On this lattice, our system consists of N polymers of fixed length. A polymer $(v_1, \dots, v_L)$ is a self-avoiding path of length $L \geq 1$, the length being the number of vertices. Moreover, the same physical constraints that impose the polymers to be self-avoiding impose that two different polymers cannot intersect. We denote by $\mathcal{S}$ the state space of all the possible configurations, so $\mathcal{S}$ can be written:

$$\mathcal{S} = \{(C_k), k = 1 \dots N, C_k \text{ is a self-avoiding path of length } L \text{ and } C_i \cap C_j = \emptyset \text{ for } i \neq j\}$$

Figure 1 shows a particular state of a system with $N = 100$ polymers of length $L = 25$ each and in dimension $d = 2$, so that the underlying lattice is actually a torus. Note that some polymers, which are in one piece on the torus, are disconnected in such a plane representation.

When $L = 1$, polymers are reduced to a vertex; for $L = 2$, the polymers are actually called dimers. A fair amount of literature exists on the so-called dimer-monomer problem, but none of them covers our case. There is a total of a^d vertices, and the polymers occupy NL vertices: it is easy to see that as soon as $NL \leq a^d$, it is possible to cover the underlying lattice with the specified polymers. Indeed, there clearly exists a self-avoiding path of length a^d that covers $\mathcal{G}$: this is just a matter of suitably enumerating $\left(\mathbb{Z}/a\mathbb{Z}\right)^d$. Then, you may divide this path into N pieces of length L each, and you have a particular state for a system that satisfies $NL = a^d$.

Since a polymer is an undirected self-avoiding path, we can think of a polymer C as an undirected subgraph of $\mathcal{G}$. However, we will sometimes need to give an orientation to the edges. This orientation is naturally induced by choosing an endpoint of the polymer. If $C = (v_1, \dots, v_L)$, then the choice of v_1 will induce the orientation $v_1 \to v_2 \to \dots \to v_L$, while the choice of v_L induces $v_L \to v_{L-1} \to \dots \to v_1$.

We denote by $q(\cdot)$ the targeted probability distribution on our system. Our aim is to run a Markov chain that has q as stationary distribution. Typically, each state $S \in \mathcal{S}$ is assigned a probability $q(S) = Z^{-1}e^{-E(S)}$ where E is an energy function, and Z is the normalizing constant, also called partition function. As in the Metropolis algorithm, our probability of acceptance will depend only on ratios of values of q, thus making no need to know Z.

2.2. **Main ideas for the growth of a polymer.** The RG* algorithm tries to replace one polymer at each step. Hence the main difficulty that it has to address is the following: given $N - 1$ polymers that do not intersect each other, how do we grow a new one? In this section, we emphasize the two salient features of the RG* algorithm introduced earlier: the concepts of feeler and of underlying graph. In order to understand why these two ideas give very good results, it is helpful to have in mind the two following extreme algorithms.

The first algorithm is an exhaustive algorithm, and works as follows. First, we pick a free vertex v_1, i.e., a vertex that is not occupied by any of the $N-1$ polymers. Then we take a free vertex v_2 neighbor of v_1 in $\mathcal{G}$: we have constructed a partial

polymer (v_1, v_2) of length 2, and we can continue from v_2. Then one of the two following events happens.

Either we always have the possibility to pick up a neighboring unoccupied vertex, and we construct a polymer $(v_1, \dots, v_L)$ which has the desired length L. Then we are done.

Or at some point, we have grown a partial polymer $(v_1, \dots, v_i)$ with $1 \leq i \leq L-1$ such that v_i has no free neighbor: each neighbor either belongs to some other polymer or is one of the preceding vertices $v_1, \dots, v_{i-1}$. In this case, we recoil to v_{i-1} and consider again the partial polymer $(v_1, \dots, v_{i-1})$, but this time, we do not grow towards v_i. The same discussion applies again: either we have another acceptable direction to grow the partial polymer, or we have to recoil to v_{i-2}, ... Observe that in this example, if from v_{i-1} we try another direction $v_i' \neq v_i$ such that from v_i' we have to recoil again, then from v_{i-1}, two directions would now be forbidden, namely v_i and v_i'.

Finally, we stop either when we have grown a polymer of length L, in which case we successfully grew a polymer, or when we have to recoil from the very first vertex v_1, in which case the step was a failure. In the former case, we have randomly chosen one possible polymer starting from v_1, while in the latter case, there was no acceptable polymer starting from v_1.

This algorithm is therefore efficient in the sense that it grows successfully a polymer whenever it is possible. The chain construction rate is thus the best possible. However, due to an exhaustive search when growing the new polymer, this answer may come at the cost of very long computations, especially if Q or L is large, and if the system is dense.

At the other extreme, the second algorithm is very fast and works identically, except that it forbids recoiling. In this algorithm, we keep picking up free neighbors at random, until either we have grown an acceptable polymer, or we have no free neighbor around the extremity of the current partial polymer. This algorithm quickly terminates, because it does at most $L-1$ trials, but the chain construction rate is very low.

The concept of feeler makes it possible to tune the RG* algorithm between these two extremes. Another way to look at these two algorithms is the following: for the exhaustive algorithm, no vertex on the current partial polymer $(v_1, \dots, v_i)$, except v_1, is sure to be in the final polymer, if there is one. Indeed, it may happen that we recoil all the way back to v_1, and from v_1, try another direction $v_2' \neq v_2$. For the fast algorithm, as soon as a vertex is added to the current partial polymer, it is sure to be in the final polymer, if there is one.

In the RG* algorithm, a partial polymer $(v_1, \dots, v_i)$ can always be broken up into two parts: the first part $(v_1, \dots, v_\Delta)$ is sure to be in the potential final polymer, while the extremity $(v_{\Delta+1}, \dots, v_i)$ serves as a retractable feeler. In other words, there is an index Δ such that the partial polymer cannot recoil below v_Δ, and this index is increasing according to certain rules that we describe in details in Section 3.3. The existence of a retractable feeler makes it possible to recoil from dense areas, thus improving the chain construction rate, while the existence of this index Δ keeps the computational complexity reasonable.

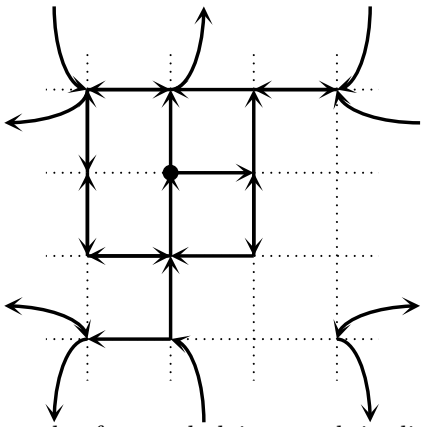


FIGURE 2. Example of an underlying graph in dimension 2 with parameter $k = 2$. The dashed lines represent the underlying lattice. We can see that each vertex has exactly $k = 2$ out-edges. The dotted vertex is the root.

It is essential to observe that the three algorithms described above do not depend on the graph they are run on. In particular, they can be run on any oriented graph. Indeed, all that matters is to know which directions are allowed to try to grow the polymer. However, even if these algorithms could be run on any graph, this graph plays an important role. For instance, we have seen that the exhaustive algorithm is intractable when Q is high, because the number of paths explored may be of order exponential in Q. But since the algorithm that generates a polymer may be run on any graph, it is therefore a very natural idea to run it on some oriented subgraph G instead of $\mathcal{G}$ itself. The graph on which the algorithm is actually run is precisely what we call the underlying graph. Since we consider subgraphs of $\mathcal{G}$, the out-degree of each vertex in an underlying graph is at most Q. The simplest underlying graphs, and the ones that we consider until Section 4.3.2, are graphs that have a constant out-degree k, meaning that each vertex of the subgraph has the same out-degree $k \leq Q$. For $k < Q$, the sets of allowable directions are actually reduced, leading to a gain in complexity. However, by forbidding some directions that could potentially lead to an admissible polymer, the construction rate is reduced as well.

Figure 2 shows an example of an underlying graph with parameter $k = 2$ in dimension $d = 2$. We see that each vertex has indeed exactly two out-edges, and some edges are curved to highlight the torus structure of the underlying lattice. Finally, one can see a dot on a vertex: this vertex is the root of the underlying graph. It is a special vertex that indicates where to begin the growth of the polymer. More details on the structure and generation of underlying graphs are given in Section 3.2.

Similarly as for ℓ, the length of the feeler, the higher k, the higher the construction rate and the higher the computational cost. So here again, a trade-off has to be made.

3. The Recoil Growth Algorithm to generate multi-polymer systems

3.1. **Overview.** With the heuristic description made in Section 2.2, we can now give an overview of the RG* algorithm. At the beginning of the algorithm, the

N polymers are placed on $\mathcal{G}$, for instance as the result of the procedure given in Section 2.1, but any other state is fine.

Assume that the system is currently in the state S_{old}, defined by the configurations of the N different polymers on the underlying lattice $\mathcal{G}$. The first step of the algorithm is to choose a polymer C_{old} uniformly at random among the N possible, and to remove it from the system. In this modified system, with only $N-1$ polymers left, we now try to grow a new polymer C_{new}. In order to do so, we first generate at random an underlying graph G_{new}, then we try to grow a new polymer C_{new} on G_{new}: this is the RG* procedure. At this point, two things may happen:

(i) the growth is not successful: then this step is a failure, and we go back to the original state S_{old} by putting back C_{old}.

(ii) the growth is successful: we now have two states S_{old} and S_{new} that only differ by one polymer, C_{old} in S_{old} and C_{new} in S_{new}. According to the Metropolis algorithm, the right stationary distribution q is obtained by accepting the new state S_{new} with a suitable probability. But here, by conditioning the generation of C_{new} according to a graph G_{new}, we introduced a skewness between S_{old} and S_{new}. By analogy with the auxiliary variable method, we need to generate an underlying graph G_{old} that plays for S_{old} the role that G_{new} plays for S_{new}. This generation is the next step of the RG* algorithm. Then the right probability of acceptance is not the probability of accepting S_{new} compared to S_{old}, but the probability to accept $(S_{\text{new}}, G_{\text{new}})$ compared to $(S_{\text{old}}, G_{\text{old}})$ that we denote by $P_{\text{acc}}\big((S_{\text{old}}, G_{\text{old}}), (S_{\text{new}}, G_{\text{new}})\big)$. To compute this probability, we first need to compute the weights of C_{old} and C_{new} on G_{old} and G_{new} respectively. Weights are complex objects that are defined in Section 3.3. Because they are directly connected to the probability $P_{\text{g}}(C|G)$ of generating the polymer C on the underlying graph G, they are nonetheless necessary in order to compute the right probability of acceptance.

For the sake of notations, in the remainder of this paper, the subscripts old and new will be noted o and n respectively, so for instance C_{o} denotes C_{old}.

One step of the RG* algorithm can be summarized as follows:

---

1. Choose a polymer C_{o} w.p. $1/N$ and remove it from the system
2. Generate at random an underlying graph G_{n} w.p. $P_{\text{u}}(G_{\text{n}})$
3. Run the RG* procedure on G_{n}
   (a) If the procedure did not successfully grow a new polymer:
      i. Put back C_{o}
      ii. Go to 1.
   (b) Else, we have grown a new polymer C_{n} w.p. $P_{\text{g}}(C_{\text{n}}|G_{\text{n}})$:
      i. Generate at random an underlying graph G_{o} compatible with C_{o} w.p. $P_{\text{c}}(G_{\text{o}}|C_{\text{o}})$
      ii. Compute the weights $W(C_{\text{n}}|G_{\text{n}})$ and $W(C_{\text{o}}|G_{\text{o}})$
      iii. Set $p = P_{\text{acc}}\big((S_{\text{o}}, G_{\text{o}}), (S_{\text{n}}, G_{\text{n}})\big)$, flip a p-coin:
         (A) If heads, add C_{n} to the polymer system
         (B) If tails, put back C_{o}
      iv. Go to 1.

With this description of our algorithm, it is straightforward to write down the transition probability $P(S_{\rm o} \to S_{\rm n})$ from $S_{\rm o}$ to $S_{\rm n}$ where $S_{\rm o}$ and $S_{\rm n}$ differ by only one polymer $C_{\rm o}$ in $S_{\rm o}$ and $C_{\rm n}$ in $S_{\rm n}$:

$$(1)\quad P(S_{\rm o} \to S_{\rm n}) = \frac{1}{N} \sum_{G_{\rm n},\ G_{\rm o}} P_{\rm u}(G_{\rm n}) P_{\rm g}(C_{\rm n}|G_{\rm n}) P_{\rm c}(G_{\rm o}|C_{\rm o}) P_{\rm acc}\big((S_{\rm o}, G_{\rm o}), (S_{\rm n}, G_{\rm n})\big).$$

Each term of this sum will be computed below.

A complete description of the RG* algorithm, to be given next, is as follows: (i) define precisely the concept of underlying graph as well as the notion of compatibility between an underlying graph and a polymer; (ii) explain the RG* procedure, in which the concept of feeler plays a central role, and finally (iii) define the weight of a polymer on an underlying graph, which makes it possible to to compute the right probability of acceptance.

3.2. **Underlying graphs.**

Definition 1. *An underlying graph $G \subset \mathcal{G}$ is a directed subgraph of $\mathcal{G}$ rooted at a vertex r such that:*

1. *for any vertex $v \in G\backslash\{r\}$, there exists a directed path from r to v*
2. *each vertex has exactly k out-edges*

Remark: Several vertices may satisfy the same property as the root, namely accessibility to all other vertices. However, the root plays a special role, as it will be the starting vertex for the growth of a polymer.

As mentioned earlier, $1 \leq k \leq Q$ is a parameter of the algorithm. Note that when $k = Q$, every underlying graph spans the whole underlying lattice with every possible directed edge induced by $\mathcal{G}$. So in this case, an underlying graph is simply characterized by its root. When $k = 1$, an underlying graph is just an oriented path starting at the root, and ending as soon as a loop appears. Figure 2 shows an example of underlying graph with $k = 2$ in dimension 2. The dotted vertex is the root, and one can check that there exists a path from the root to any vertex of the underlying graph.

Definition 2. *We say that a polymer C and an underlying graph G are* compatible *if the root r of G is one of the two extremities of C and if $C \subset G$ (where the choice of r induces the orientation on C as explained in Section 2.1).*

The class of underlying graphs is the natural class of graphs to run the RG* procedure; as we will see, a polymer can be grown on some underlying graph if and only if they are compatible.

3.2.1. *Procedure to generate an underlying graph.* After choosing a random polymer to be removed from the system, the first step of the RG* algorithm is to generate an underlying graph. The growth of the candidate polymer will be performed on this underlying graph.

To generate an underlying graph, we proceed iteratively, taking care that every vertex added to the graph has exactly k out-edges. This algorithm is basically a greedy algorithm that at each step assigns out-edges to some randomly chosen vertex.

More precisely, we have two sets V and T: V is the set of vertices of the final underlying graph that we are trying to generate, whereas T is a waiting room for the vertices that will eventually end up in V, but have not been assigned out-edges yet.

Initially, we choose the root r uniformly at random in the set of vertices that are not occupied by any of the $N-1$ polymers in the system. Thus $T = \{r\}$ and $V = \emptyset$ because we have not assigned out-edges to the root yet. The first step is to assign k out-edges to r: r has Q neighbors in $\mathcal{G}$, and we choose k out-edges out of these Q possible uniformly at random. Call $v_1, \dots, v_k$ the corresponding neighbors or r. Then we have assigned out-edges to r but not to the k new vertices, so $T = \{v_1, \dots, v_k\}$ and $V = \{r\}$. Then we take any vertex in T, say v_1, and we choose its k neighbors. Since r and v_1 are neighbors in $\mathcal{G}$, it may happen that r is among these neighbors, so we have to be careful: we add to T the vertices that are not already in V or in T, and we move v_1 from T to V. We keep on assigning out-edges to vertices in T until $T = \emptyset$.

We see that in this algorithm, except for the root, the current configuration of the system is not taken into account. We add vertices and out-edges independently of the polymers on $\mathcal{G}$. This leads to some inefficiency, and we explain in Section 5 how to remedy to this problem in practice.

When T is empty, it is clear that each vertex in V has exactly k out-edges, and that there exists a directed path from r to any vertex: we have indeed generated an underlying graph. Observe that there is no constraint on the in-degree of each vertex. However, since there is a path from the root to any vertex but the root, each vertex except the root has in-degree at least 1. The root is the only vertex that may have in-degree 0.

In the following pseudo-code that explains how to generate an underlying graph, E is the set of edges we are trying to generate, and V and T are as in the above informal description:

---

1. Pick a free vertex r at random, and initialize as follows:
   (a) $V = E = \emptyset$
   (b) $T = \{r\}$
2. While $T \neq \emptyset$:
   (a) Pick any vertex v in T
   (b) Choose k different neighbors $(v_1, \dots, v_k)$ of v uniformly at random
   (c) For $i = 1 \dots k$
      i. Add the edge $v \to v_i$ to E
      ii. If $v_i \notin V$ and $v_i \notin T$, add v_i to T
   (d) Remove v from T and add it to V

Since the underlying lattice $\mathcal{G}$ is finite, it is clear that this procedure terminates in finite time. Moreover, it is easy to see that this procedure does not depend on the order in which you pick up vertices in T: once a vertex is in T, its set of out-edges does not depend on when you assign them. Figure 3 page 12 shows the six first steps of the generation of an underlying graph in dimension two with $k = 2$.

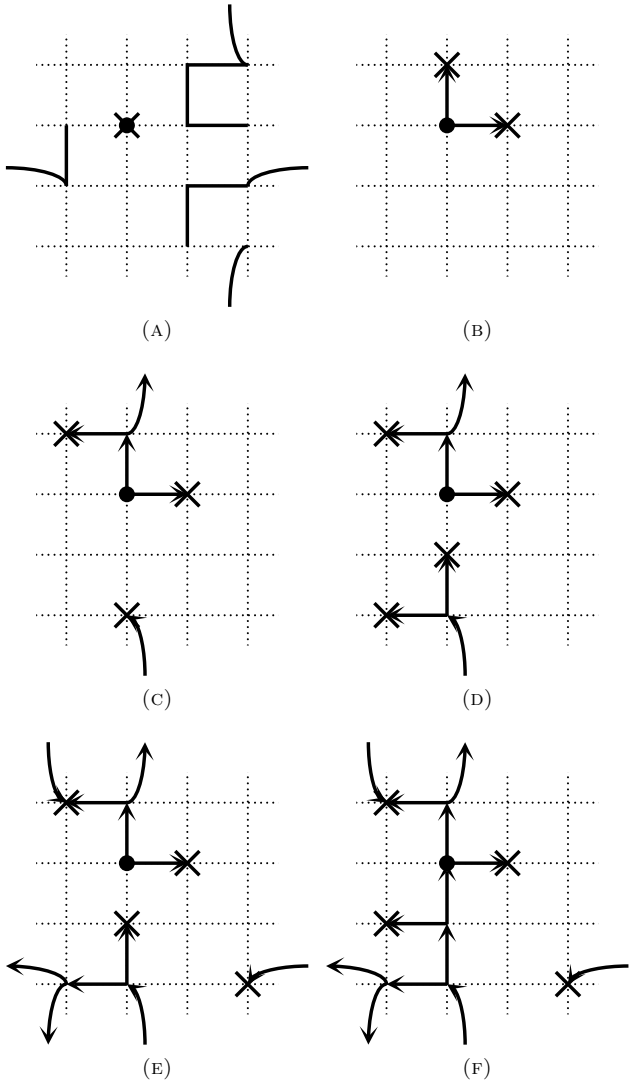


FIGURE 3. The six first steps of the generation of an underlying graph. (a): the root r is the dotted vertex, which we pick uniformly at random among the unoccupied vertices. The edges represent the $N-1=2$ polymers left in the system. In the sequel of the generation of the underlying graph, the polymers play no role, so they are not drawn. (a)-(f): vertices with a cross will be in the final underlying graph, but have not been assigned out-edges to yet: they form the set T. At each step, we pick one vertex in T at random, and assign out-edges to it. The vertex then becomes part of the set V, and we remove the cross. If by assigning out-edges, we have discovered new vertices, we add these vertices to T. In (f), the generation is not over, since T has four more vertices.

Proposition 1. *The probability $P_{\mathrm{u}}(G)$ to generate the underlying graph G is given by*

$$P_{\mathrm{u}}(G) = \frac{\alpha^{|G|}}{\gamma} \tag{2}$$

where $\alpha = \binom{Q}{k}^{-1}$, $|G|$ is the number of vertices in G, and $\gamma = a^d - (N-1)L$ is the number of free vertices when $N-1$ polymers are in the system.

Proof. The generation of G is unambiguous: the root is chosen with probability $1/\gamma$; then, each time we assign out-edges, we have to assign the right set of out-edges. Since there are $1/\alpha$ such sets and the choice is uniform random, the probability of assigning the right set of out-edges is exactly α. Such a choice occurs for each of the $|G|$ vertices, hence the result. □

3.2.2. *Procedure to generate a compatible underlying graph.* As we have seen in the overview of the RG* algorithm in Section 3.1, the transition between the old and the new state is made symmetric by generating an underlying graph compatible with the old polymer C_{o}, as a comparison to the graph G_{n}.

The procedure to do so is essentially the same as to generate any underlying graph: the only difference is that when we try to assign out-edges to vertices that belong to $C_{\mathrm{o}} = (v_1, \ldots, v_L)$, we must take care that for each $1 \leq i \leq L-1$, we do have $v_i \to v_{i+1}$ in the set of out-edges. Hence for these $L-1$ vertices, one out-edge is imposed, and we have to choose the $k-1$ others among $Q-1$ possible. Hence we can derive the following property similarly as for general underlying graphs:

Proposition 2. *Given a polymer C of length L, the probability $P_{\mathrm{c}}(G|C)$ to generate an underlying graph G compatible with C is given by*

$$P_{\mathrm{c}}(G|C) = \frac{\alpha^{|G|-L+1}\beta^{L-1}}{2} \tag{3}$$

where $\beta = \binom{Q-1}{k-1}^{-1}$ and α and $|G|$ are as in Property 1.

Remark: $P_{\mathrm{c}}(G|C)$ and $P_{\mathrm{u}}(G)$ are proportional: we have $P_{\mathrm{c}}(G|C) = \eta P_{\mathrm{u}}(G)$ where $\eta = \gamma/2 \cdot (\beta/\alpha)^{L-1}$ is a universal constant that does not depend on the polymer or the underlying graph.

3.3. **Recoil Growth Procedure.** In this section, we describe in details the procedure that grows a polymer on a given underlying graph G. As explained before, this growth takes place on an underlying graph that has a constant out-degree k. Before describing the dynamics according to which the algorithm either extends an existing partial polymer or recoils, we have to understand the concept of feeler.

3.3.1. *Decomposition of a partial polymer $\overline{C}$.* During the growing process, a partial polymer $\overline{C} = (v_1, \ldots, v_i)$ of length $i < L$ can always be decomposed into two disjoint parts: the *fixed part* and the *feeler*. Before stating the definition of this decomposition, it is important to have in mind the rule that this decomposition stands for:

Rule: *During the growth process, we are allowed to recoil from a vertex if and only if it is in the feeler.*

This means that if at some point we need to recoil from a vertex that belongs to the fixed part, the attempt is a failure. Since the feeler is the complementary part of the fixed part, one only needs to define the fixed part:

Definition 3. *A node v_j is in the fixed part if $j = 1$ or if during the history of the growth of $\overline{C}$, there was a partial polymer $(v_1, \dots, v_j, v_{j+1}, \dots, v_{j+\ell})$.*

Equivalently, the fixed part is the set of vertices v such that either v is the starting vertex of the polymer, or at some point in the growth history, a partial polymer has been grown through v and has been ℓ steps further. The previous definition immediately yields the following properties:

Proposition 3. *There exists an increasing index Δ such that the fixed part is of the form $(v_1, \dots, v_\Delta)$.*

Proof. Consider a vertex v_j in the fixed part, and the corresponding partial polymer $\overline{C} = (v_1, \dots, v_j, v_{j+1}, \dots, v_{j+\ell})$. Since the RG* procedure extends or shortens a partial polymer only one vertex at a time, this implies that necessarily, at some point, the shorter polymer $(v_1, \dots, v_{j-1}, v_j, \dots, v_{j+\ell-1})$ was grown: this exactly means that $j - 1$ is in the fixed part as well. This proves that the fixed part is indeed of the form $(v_1, \dots, v_\Delta)$.

Moreover, it is clear by definition that if a vertex belongs at some point to the fixed part, then it stays in the fixed part for the remainder of the growth process: the fixed part is increasing, hence so is the index Δ. □

Proposition 4. *The feeler is of the form $(v_{\Delta+1}, \dots, v_i)$, and is of length at most ℓ (and is possibly empty).*

Proof. By the previous proposition, all we have to show is that the feeler is indeed of length at most ℓ. Imagine for a moment that the feeler is of length $\ell + 1$: this means that we have a partial polymer $\overline{C} = (v_1, \dots, v_\Delta, v_{\Delta+1}, v_{\Delta+2}, \dots, v_{\Delta+\ell+1})$ where Δ delimits the fixed part. But then, by definition of the fixed part, the vertex $\Delta + 1$ should belong to the fixed part, which is not possible. □

We can now draw a picture of the way the fixed part and the feeler evolve. If $\overline{C} = (v_1, \dots, v_i)$ grows towards v_{i+1}, the feeler is increased except when it was already of length ℓ. If the feeler is of length ℓ when $\overline{C}$ grows, then it stays of length ℓ, and the fixed part grows. It is then straightforward to derive the formula $\Delta = \max(1, L_{\max} - \ell)$, where $L_{\max}$ is the length of the longest polymer grown in the history of C.

Note that vertices v_j with $j \geq L - \ell$ are never in the fixed part, because we never grow a polymer of length larger than L.

By playing on ℓ, we can have the two extreme algorithms described in Section 2.2. For $\ell = 0$, we get the fast algorithm, because since the feeler is of length 0, there is no vertex from which it is allowed to recoil. And if one sets $\ell = L$, it is always allowed to recoil from any vertex, and we get the exhaustive algorithm. So this parameter is the parameter that makes it possible to make a trade-off between the speed and the tractability of the algorithm.

We can now fairly simply describe the RG* procedure given an underlying graph G.

3.3.2. *Description of the RG* procedure.* Given an underlying graph G, the RG* procedure keeps record of the current partial polymer $\overline{C}$, the length $L_{\max}$ of the longest partial polymer ever grown, and L sets D_i. For a partial polymer of length at least i, D_i represents the set of vertices that the vertex v_i has already tried as directions of growth. In particular, at any time, D_i is a subset of the set of neighbors of v_i in G. These sets are here to insure that the algorithm is free of loop: when you recoil to a vertex v_i, you must try a new direction, that you have not tried so far. If at some point, you recoil to v_i and every neighbor of v_i is in D_i (equivalently, $|D_i| = k$), this means that you have again to recoil from v_i. This is possible only when v_i is in the feeler, which is easily verified by checking the condition $i > \max(1, L_{\max} - \ell)$.

The RG* procedure works as follows:

---

1. Initialization:
   (a) $\overline{C} = (v_1)$, where v_1 is the root of G
   (b) $D_1 = \emptyset$
   (c) $L_{\max} = 1$
2. At each step, with $\overline{C} = (v_1, \dots, v_i)$:
   (a) If $i = L$, stop, $\overline{C}$ is a complete polymer
   (b) Else if $|D_i| = k$, recoil:
      i. If $i > \max(1, L_{\max} - \ell)$, set $\overline{C} = (v_1, \dots, v_{i-1})$ and go to 2
      ii. Otherwise, stop, the generation has failed
   (c) Else, keep picking uniformly at random a vertex v neighbor of v_i in G and not in D_i, add it to D_i, and stop when either v is unoccupied or $|D_i| = k$:
      i. If v is unoccupied, set $\overline{C} = (v_1, \dots, v_i, v)$, $D_{i+1} = \emptyset$, update $L_{\max}$ and go to 2
      ii. Else, $|D_i| = k$, and recoil as specified in 2.(b)

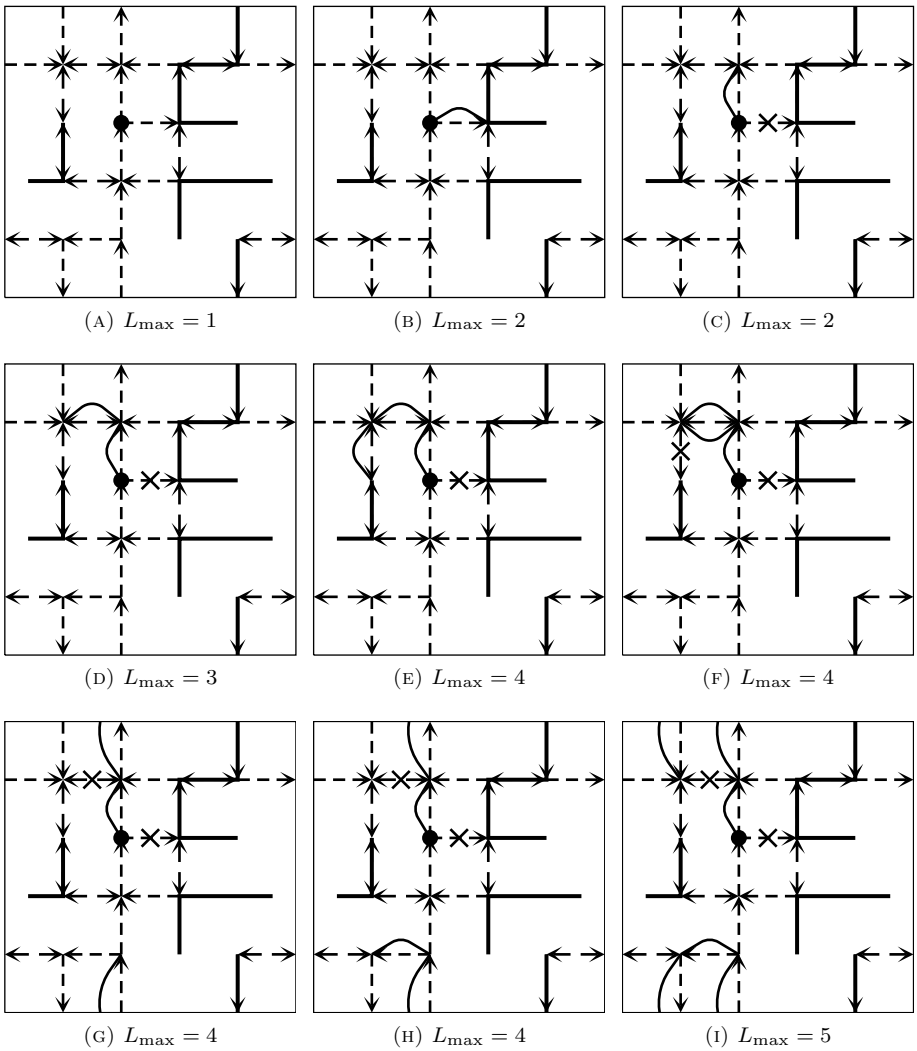
It is not hard to show that this procedure is free of loop, and therefore terminates in finite time. The absence of loops is insured by the sets D_i: you cannot grow twice the same polymer because this would imply to choose as next direction a vertex that already belongs to some set D_i.

Figure 4 on page 16 shows the successful generation of a polymer in dimension $d = 2$, with parameters $N = 3$, $L = 5$, $k = 2$ and $\ell = 2$.

With this description, it is clear that if the RG* procedure returns a polymer C, then C and G are compatible. Hence the following property holds:

Proposition 5. *If C and G are not compatible, then $P_{\mathrm{g}}(C|G) = 0$.*

We still have to compute the probability $P_{\mathrm{g}}(C|G)$ when C and G are compatible. To get an hindsight of what the right answer is, let us consider the case $\ell = 0$: write $C = (v_1, \dots, v_L)$, and consider any $1 \leq i \leq L - 1$. In the process of growing C, we have necessarily grown at some point the partial polymer $(v_1, \dots, v_i)$, and from v_i, we have k choices. But since $\ell = 0$, any choice different than v_{i+1} will not lead to C. Indeed, if we choose $v' \neq v_{i+1}$, since $\ell = 0$, we will never recoil below v', therefore we will not grow C. In opposition, if we choose v_{i+1} for each $1 \leq i \leq L - 1$, we will



(A) $L_{\max} = 1$ (B) $L_{\max} = 2$ (C) $L_{\max} = 2$

(D) $L_{\max} = 3$ (E) $L_{\max} = 4$ (F) $L_{\max} = 4$

(G) $L_{\max} = 4$ (H) $L_{\max} = 4$ (I) $L_{\max} = 5$

FIGURE 4. Example of the growth of a polymer with parameters $d = 2$, $N = 3$, $L = 5$, $k = 2$ and $\ell = 2$. The 2 polymers in the system are represented by the bold edges, the dashed arrows represent the underlying graph. (a): we start from the root. (b): we try the neighbor to the right, and we update the counter $L_{\max}$ to 2. This vertex is occupied by another polymer, so we recoil to the root. Since $\ell > 0$, this move is allowed. (c): we try another direction, and go up, which is the only direction left possible since the cross symbolizes the set D_1 of directions already tried. (d): we try the left vertex, and update $L_{\max}$. (e): we try the down vertex, and update $L_{\max}$. This vertex is occupied by another polymer, so we recoil. (f): we try the right vertex, which is occupied by the current polymer. Since this is the second try and $k = 2$, we have to recoil again. Since $\ell > 1$, this is admissible. If $\ell = 1$, this step would yield a failure, because $\max(1, L_{\max} - 1) = 3$, and we want to recoil from v_3. (g)-(i): we try admissible neighbors that lead to a full polymer.

grow C: this happens with probability $(1/k)^{L-1}$, which is the answer for $\ell = 0$.

In the case $\ell > 0$, we have still necessarily grown at some point the partial polymer $(v_1, \dots, v_i)$, and from v_i, there are still k potential choices. But in this case, if we choose a vertex $v \neq v_{i+1}$, this is not necessarily the end, because it may happen that we recoil back from v to v_i. In this case, and when $k > 1$, we then have a second opportunity to pick the right vertex v_{i+1}.

Since $\ell > 0$, when we grow from v_i to v, v is in the feeler part, and we will eventually recoil back to v_i if and only if v never belongs to the fixed part. Since by definition, v belongs to the fixed part if and only if at some point, we grow the chain ℓ steps further, we get that we will recoil back to v_i if and only if the partial polymer $(v_1, \dots, v_i, v)$ cannot be extended ℓ steps further on G. Note that this equivalence is true only for vertices that potentially may belong to the fixed part, i.e., for vertices v_i with $i \leq L - \ell$.

For $i \geq L - \ell + 1$, this equivalence becomes: we will recoil back to v_i if and only if the partial polymer $(v_1, \dots, v_i, v)$ cannot be completed.

Definition 4. *For $1 \leq i \leq L-1$, a neighbor v of v_i is called admissible if either $i < L - \ell$ and the partial polymer $(v_1, \dots, v_i, v)$ can be extended ℓ steps further, or if $i \geq L - \ell$ and $(v_1, \dots, v_i, v)$ can be completed to an entire polymer.*
The elementary weight w_i of v_i is the number of admissible neighbors of v_i.

Remark: In this definition, when the partial polymer $(v_1, \dots, v_i, v)$ is extended, possible interactions with the remaining part $(v_{i+1}, \dots, v_L)$ of the polymer are not taken into account. Moreover, note that when C and G are compatible, v_{i+1} is always admissible, because we know that $(v_1, \dots, v_L) = C$ is possible. Hence in this case, we always have $w_i \geq 1$. And now we can compute the probability $P_{\mathrm{g}}(C|G)$:

Definition 5. *$W(C|G) = \prod_1^{L-1} w_i$ is the* weight *of the polymer C given the underlying graph G.*

Proposition 6. *Given the underlying graph G compatible with C, we have*

$$P_{\mathrm{g}}(C|G) = \frac{1}{W(C|G)} = \left(\prod_{i=1}^{L-1} w_i \right)^{-1}. \tag{4}$$

Proof. Let $i \in \{1, \dots, L-1\}$. In the process of growing C, the partial polymer was at some point $(v_1, \dots, v_i)$. By definition, if we choose as next vertex v, then we will eventually recoil back to v_i if and only if v is not admissible. Hence the probability from v_i to make the right choice is one over the number of admissible neighbors, i.e., $1/w_i$. □

Remark: To compute the elementary weight w_i, we have no choice but to use the exhaustive algorithm presented previously. Indeed, we need to know whether under certain constraints, we can grow a self-avoiding path of length ℓ. So if ℓ is large, since we have to do this at most $(k-1)(L-1)$ (when for each $1 \leq i \leq L-1$, we must try to grow a feeler from each of its $k-1$ neighbors) computing the weight can be expensive. However, this is only to be done when we have successfully grown a polymer. The complexity here is linear in k and L, but exponential in ℓ.

4. Properties of the Recoil Growth Algorithm

Definition 6. *The density $\rho = NL/a^d$ of the system is the ratio of the number of vertices occupied over the total number of vertices.*

In this section, we prove the following theorem, where it is assumed that the size a of the underlying lattice satisfies $a \geq L$, which is the relevant case in practice:

Theorem 1. *With the right probability of acceptance $P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big)$, the RG* algorithm is reversible with stationary distribution q. The algorithm is moreover irreducible at densities smaller than $\lfloor a/L \rfloor / a$.*

To prove this theorem, we first deal with the stationary distribution in the next section, and then discuss some issues related to the irreducibility. Note that we prove the existence of a non-empty region for which both the original RG and the RG* algorithms are irreducible.

4.1. **Reversibility.**

Lemma 1. *Take any two states S_{o} and S_{n} in $\mathcal{S}$ that differ by only one polymer C_{o} in S_{o} and C_{n} in S_{n}, and take any two underlying graphs G_{o} and G_{n} compatible with C_{o} and C_{n} respectively. If we choose*

$$P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big) = \min\left(1, \frac{q(S_{\mathrm{n}})W(C_{\mathrm{n}}|G_{\mathrm{n}})}{q(S_{\mathrm{o}})W(C_{\mathrm{o}}|G_{\mathrm{o}})}\right), \tag{5}$$

then q satisfies the detailed balance equations:

$$q(S_{\mathrm{o}})P(S_{\mathrm{o}} \to S_{\mathrm{n}}) = q(S_{\mathrm{n}})P(S_{\mathrm{n}} \to S_{\mathrm{o}}).$$

Proof. Recall that we have from (1)

$$P(S_{\mathrm{o}} \to S_{\mathrm{n}}) = \frac{1}{N} \sum_{G_{\mathrm{n}},\ G_{\mathrm{o}}} P_{\mathrm{u}}(G_{\mathrm{n}})P_{\mathrm{g}}(C_{\mathrm{n}}|G_{\mathrm{n}})P_{\mathrm{c}}(G_{\mathrm{o}}|C_{\mathrm{o}})P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big).$$

We can show that q not only satisfies the detailed balance equations, but satisfies a stronger "local" condition: if we define

$$P\big((S_{\mathrm{o}}, G_{\mathrm{o}}) \to (S_{\mathrm{n}}, G_{\mathrm{n}})\big) = P_{\mathrm{u}}(G_{\mathrm{n}})P_{\mathrm{g}}(C_{\mathrm{n}}|G_{\mathrm{n}})P_{\mathrm{c}}(G_{\mathrm{o}}|C_{\mathrm{o}})P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big),$$

then $P(S_{\mathrm{o}} \to S_{\mathrm{n}}) = 1/N \sum_{G_{\mathrm{n}},\ G_{\mathrm{o}}} P\big((S_{\mathrm{o}}, G_{\mathrm{o}}) \to (S_{\mathrm{n}}, G_{\mathrm{n}})\big)$, and it is clearly sufficient to show that for all $S_{\mathrm{o}}, S_{\mathrm{n}}, G_{\mathrm{o}}$ and G_{n}, q satisfies

$$q(S_{\mathrm{o}})P\big((S_{\mathrm{o}}, G_{\mathrm{o}}) \to (S_{\mathrm{n}}, G_{\mathrm{n}})\big) = q(S_{\mathrm{n}})P\big((S_{\mathrm{n}}, G_{\mathrm{n}}) \to (S_{\mathrm{o}}, G_{\mathrm{o}})\big). \tag{6}$$

But we have seen that $P_{\mathrm{u}}(G_{\mathrm{n}})P_{\mathrm{c}}(G_{\mathrm{o}}|C_{\mathrm{o}}) = P_{\mathrm{u}}(G_{\mathrm{o}})P_{\mathrm{c}}(G_{\mathrm{n}}|C_{\mathrm{n}})$ because there exists a constant η such that $P_{\mathrm{c}}(G|C) = \eta P_{\mathrm{u}}(G)$, so (6) amounts to

$$\frac{P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big)}{P_{\mathrm{acc}}\big((S_{\mathrm{n}}, G_{\mathrm{n}}), (S_{\mathrm{o}}, G_{\mathrm{o}})\big)} = \frac{q(S_{\mathrm{n}})W(C_{\mathrm{n}}|G_{\mathrm{n}})}{q(S_{\mathrm{o}})W(C_{\mathrm{o}}|G_{\mathrm{o}})},$$

what is true by choice of the probability of acceptance. □

4.2. **On the irreducibility of the Markov Chain.**

Knowing that q satisfies the detailed balance equations is not enough for the algorithm to work properly: one needs results on the irreducibility of the Markov Chain as well. This question seems to be a hard problem, and we aim at giving some hints in this section.

4.2.1. *Proof of theorem 1.*

Lemma 2. *For any d, N, L and $a \geq L$, the chain is irreducible if $\rho \leq \lfloor a/L \rfloor / a$.*

Proof. The idea is to show that from any configuration $S \in \mathcal{S}$, we can reach a particular configuration $\widetilde{S}$ where all the polymers are straight, and lie in some specific boxes. The boxes are defined as follows.

$\mathcal{G} = (\mathbb{Z}/a\mathbb{Z})^d$ is a d-dimensional cube, and for any $x \in (\mathbb{Z}/a\mathbb{Z})^{d-1}$, we consider the subset $F_x = \{y = (y_i) \in \mathcal{G} : (y_1, \ldots, y_{d-1}) = x, y_d \in a\mathbb{Z}\}$ where the $d-1$ first coordinates are fixed. F_x is in bijection with $\mathbb{Z}/a\mathbb{Z}$. Since $a \geq L$, we write $a = nL + r$ with $n = \lfloor a/L \rfloor \geq 1$ and $0 \leq r \leq L-1$. For each x, we can then divide F_x into n boxes, the first box being in bijection with $\{0, \ldots, L-1\}$, the second with $\{L, \ldots, 2L-1\}$, etc... The total number of boxes B is equal to $a^{d-1}n$, therefore we get, using the definition of ρ and the hypotheses:

$$\frac{B}{NL} = \frac{n}{\rho a} \geq 1.$$

But NL is the number of occupied vertices, so the previous inequality means that we have more boxes than occupied vertices. We study only the worst case, where $B = NL$: one of two things may happen.

If each box is occupied by exactly one vertex, then each box is intersected by exactly one polymer. So we can consider each polymer in turn, and put each one in some box that it occupies. Each polymer will then be straight and lie in some box.

If some boxes are occupied by more than one vertex, then necessarily, some boxes are empty. Then we can consider the polymers in turn and assign them to empty boxes while we have empty boxes. We stop either when we have no more empty boxes or when every polymer has been assigned to a box. In the latter case, we are done. And it is easy to see that the former case never happens, because by putting a polymer in a box, we create empty boxes.

In any case, we can go to a state where all the polymers are in boxes. Such states are clearly connected, and the chain is therefore irreducible. □

Remark: With regards to simulation, the longer the polymers, the better. In [3], the authors investigate cases when $L = 100$. So this result insures irreducibility at very low densities, around 0.01, compared to the values we would be interested in, like 0.6. Moreover, one question that is still open at this point is to give a lower bound independent of the parameters of the system. For instance, it seems a reasonable bet that the algorithm is always irreducible for $\rho = 10^{-10}$.

4.2.2. *Informal discussion.* In this section, we give examples of parameters for which the algorithm is irreducible, some for which it is not, and we state some conjectures.

First, observe that if two states S and S' differ by only one polymer, then the probability $P(S \to S')$ of going from S to S' in one step of the algorithm is strictly positive for any k and ℓ. These two parameters obviously influence the value of such a probability, but it will nevertheless always be positive. The question of irreducibility does therefore not depend on these two parameters.

For purposes of irreducibility, the algorithm is then characterized by N, L and $\mathcal{G}$. Recall that $\mathcal{G} = (\mathbb{Z}/a\mathbb{Z})^d$ is a cyclic cube in dimension d. Since $\rho = NL/a^d$,

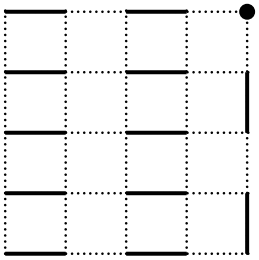


FIGURE 5. State where all the dimers are horizontal, except for one column, which contains the "whole", i.e., the free vertex.

we see that the RG* algorithm is characterized by five parameters N, L, ρ, a and d, so we adopt the notation $RG_d^*(N, L, a, \rho)$ to refer to the RG* algorithm with these parameters. Any one of these parameters is uniquely determined by the four others thanks to the relation $\rho = NL/a^d$.

Before going on, we rule out trivial cases that are of no interest: for $N = 1$ or $L = 1$, the algorithm is always irreducible. So in the rest of this discussion, we always assume $N, L \geq 2$. The case $L = 2$ corresponds to the so-called "monomer-dimer system", which has received considerable attention in the literature, e.g. [6, 14], with [6] dealing with problems close to the ones we are interested in. However, the Metropolis algorithm proposed in [6] allows to remove or add dimers, in which case the problem of irreducibility is trivial.

Proposition 7 deals with a simple case:

Proposition 7. *If $\rho = 1$, the algorithm is never irreducible.*

Proof. Assume that $\rho = 1$ and you start from some state S. Then removing a polymer C leaves exactly L free vertices. Hence for any state S' where C has been replaced by C', C and C' necessarily occupy the same set of vertices. In particular, two vertices v and v' neighbors in $\mathcal{G}$ which belong to different polymers in S will never be part of the same polymer. Since there exist such states, $RG_d^*(N, L, a, 1)$ is not irreducible. □

The next two propositions will help us build our intuition:

Proposition 8. *For any $\rho < 1$ and any other parameters, $RG_d^*(N, 2, a, \rho)$ is irreducible.*

Proof. We deal with the case $d = 2$, and when there is only one free vertex. When there are more free vertices or when $d > 2$, the same ideas can be extended to prove that the system is still irreducible. So from now on, we assume $d = 2$ and that there is one free vertex.

We show that from any state, we can reach the state depicted in Figure 5, where all the dimers are horizontal, except on one column where they all are vertical. The idea is the following: we consider the number κ of dimers that are "vertical", i.e., the number of dimers that are aligned along a column and not along a line. We will show that there exists a path, i.e., a sequence of moves, along which κ is decreasing. Figure 5 represents a state where κ is equal to its minimal value.

For the parameters considered, $a^2 = 2N + 1$, hence a is odd. Since only one vertex is free, there is only one line on which there is a free vertex. So for the $a - 1$ other lines, the a vertices are occupied by dimers. Since a is odd, and since

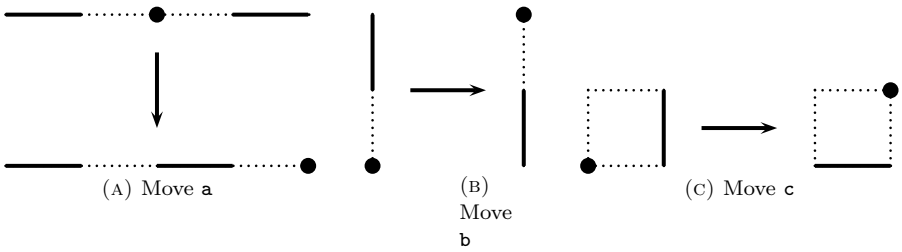


FIGURE 6. The three elementary moves: `a` and `b` do not change κ, whereas `c` decreases κ by one.

a horizontal dimer occupies an even number of vertices, there is an odd number of vertical dimers that have exactly one vertex lying on this line. In particular, there is at least one vertical dimer on each of these $a-1$ lines.

We now describe the path along which κ is decreasing. This path is made of the three elementary moves `a`, `b` and `c` depicted in Figure 6. We consider the line with the whole: there is an even number of vertices occupied, therefore there is an even number of vertical dimers sharing a vertex with this line, which is possibly null.

If there are at least two such dimers, we can move to the right thanks to move `a` until the vertex to the right of the whole is a vertical dimer, as in case (c) of Figure 6. Then by applying move `c`, we can reduce κ by one. The whole is then on a new line.

If there is no vertical dimer on the line of the whole, we can change line: first, we apply move `a` to go to the right until the whole is under some vertical dimer as in case (b). There exists such a dimer, because there is an odd number of vertical dimers on the above line. Then we apply move `b`, and we jumped two lines. Since there is an odd number of lines, by jumping two lines at a time, we can put the whole on any line (this is the same reason why you can always go under a vertical dimer).

So now, we know that from any state, we can reach a state where κ is minimum. It is not hard to see that such a state is as follows: in any line except the line with the whole, there is exactly one vertical dimer. To reach the state of Figure 5, one just has to align these $(a-1)/2$ vertical dimers: it is not hard to do so by combining moves `a` and `c`, so we are done. □

Proposition 9. *$RG_2^*(N, 5, a, 80/81)$ is not irreducible.*

Proof. We have $80/81 = 5N/a^2$, which leads by simple considerations to $N = 16n^2$ for some integer $n > 0$, and $a = 9n$. Now we restrict our attention to the case $n = 1$, in which case there are 81 vertices, so only one vertex is free. Figure 7 zooms on a region of a state of such a system, where the polymers have a specific configuration around the only free vertex. Every vertex in any of the dashed area is occupied. Hence from this state, two things may happen.

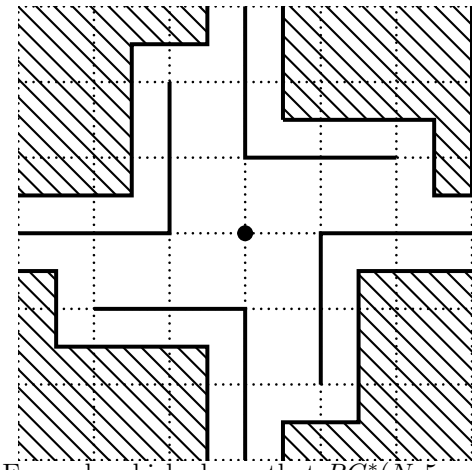


FIGURE 7. Example which shows that $RG_2^*(N, 5, a, 80/81)$ is not irreducible: there is no free vertex in the dashed areas, so polymers in the dashed areas cannot move. Nor can the polymers surrounding the only free vertex.

If we remove a polymer that is in the dashed area, the only area where we can grow another polymer is in the region freed by this polymer. Hence in this case, every polymer still occupies the same set of vertices.

But if we remove a polymer surrounding the free vertex, then it is easy to see that the only possible way to grow it is at the exact same place. Indeed, simple considerations show that a new polymer cannot occupy this free vertex, because it acts like a dead-end: from this vertex, the polymer would have to pick one of two possible directions, and whatever the direction chosen, there is not enough room to grow the polymer.

Hence from this state, the only states reachable are states where each polymer occupies the same set of vertices, so the system is not irreducible.

For $n > 1$, we can repeat over space the state obtained in the case $n = 1$ to obtain a state that has the exact same property, hence the proposition is proved. □

From these two examples, we see that the parameter ρ is not sufficient to characterize the irreducibility of the system, meaning that it can happen that $RG_d^*(N, L, a, \rho)$ is irreducible, whereas $RG_d^*(N', L', a', \rho')$ with $\rho' < \rho$ is not. However, it seems intuitively that the critical parameter is actually the length of the polymers, and so one could wonder whether for fixed length, and for fixed dimension d, the density is not enough to characterize the irreducibility. This idea is the object of the following conjecture:

Conjecture 1. *If $RG_d^*(N, L, a, \rho)$ is irreducible, then $RG_d^*(N', L, a', \rho')$ is irreducible for any N', a' such that $\rho' < \rho$.*

If this conjecture were to be true, then one could consider the quantity

$$\rho_d(L) = \sup_{N,a} \{\rho : RG_d^*(N, L, a, \rho) \text{ is irreducible}\}.$$

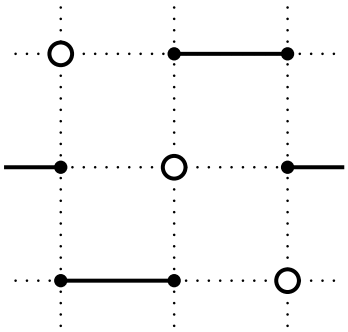


FIGURE 8. Example of a state in $RG_2^*(N, L, a, \rho)$ that cannot be extended to $RG_2^*(N+1, L, a, \rho)$, although $RG_2^*(N+1, L, a, \rho)$ is irreducible ($N = 3, L = 2$ and $a = 3$). Circles represent unoccupied vertices.

For any N, a, $RG_d^*(N, L, a, \rho)$ would be irreducible if and only if $\rho < \rho_d(L)$, and a second conjecture, that would reflect the fact that the longer the polymers, the harder it is for the system to be irreducible, would be:

Conjecture 2. *$\rho_d(L)$ is decreasing in L.*

Finally, we state the the most natural, and probably easiest, conjecture: it says that if the system is irreducible, then removing polymers, shortening the polymers or growing the box should leave the system irreducible.

Conjecture 3. *Suppose that $RG_d^*(N, L, a, \rho)$ is irreducible. Then $RG_d^*(N', L', a', \rho')$ is irreducible for any $N' \leq N, L' \leq L$ and $a \geq a'$, where the value of ρ' is determined by the other parameters.*

Note that this conjecture would immediately imply Conjecture 2. A natural idea to try to prove this last conjecture is the following: suppose for instance that you want to show that $RG_d^*(N-1, L, a, \rho')$ is irreducible, given that $RG_d^*(N, L, a, \rho)$ is. Then you could add a fake polymer, make any move in $RG_d^*(N, L, a, \rho)$, remove the fake polymer, and you are done. However, it is not always possible to add a polymer to $RG_d^*(N-1, L, a, \rho')$, even if $RG_d^*(N, L, a, \rho)$ is irreducible, as shown in the simple example on Figure 8. Hence, Conjecture 3 would be true if from any state in $RG_d^*(N-1, L, a, \rho)$, we could reach a state where we can add a fake polymer. Similar ideas hold for L and a.

4.3. **Extensions of the Recoil Growth Algorithm.**

4.3.1. *Polymers of different lengths.* All what we have done still holds when the polymers have different lengths, under the reasonable condition that we always try to replace a polymer by another polymer with the same length. Then it is easy to check that all the above formulas still hold, where each time L is to be replaced by the length L_C of the polymer C that we are trying to replace, and the constant γ is to be replaced by a number γ_{L_C}.

When the polymers have different length, a new problem arises which we have not mentioned so far: the polymers are then *de facto* distinguishable. Distinguishability influences the irreducibility of the algorithm. A trivial example is when $L = 1$ and $\rho = 1$, i.e., the full lattice is covered by its N vertices as N polymers: if the polymers are indistinguishable, the system is irreducible, because there is only one state. But if they are distinguishable, then the system is not irreducible, because there are $N!$ isolated states.

4.3.2. *Underlying graphs with random out-degrees.* In this section, we study an extension proposed in [3] that relaxes the constraint that k needs to be integer-valued, and adopt it in our framework. Although for the original RG algorithm, this does not require to change the probability of acceptance P_{acc}, for the RG* algorithm, it does. We compute below the correct value for this new algorithm, to which we refer as the *extended RG* algorithm*.

In this extension, the out-degree of each vertex of an underlying graph is random instead of deterministic, so the definition of an underlying graph given earlier does not apply in this section. However, the generation of underlying graphs, compatible or not, is very similar as in the case where the out-degrees are deterministic. There is only one additional step: instead of assigning k or $k-1$ uniformly at random, we first draw a number $1 \leq \kappa \leq Q$ with probability p_κ, and then we pick κ or $\kappa - 1$ vertices uniformly at random. The probability distribution p is therefore an additional parameter to the algorithm. Note that we impose $p_0 = 0$, because in the process of growing a compatible underlying graph, we have to assign at least one out-edge to the vertices on the polymer.

The interest of this extension is to allow any mean random degree $< k >= \sum \kappa p_\kappa$, so that the optimization of the algorithm can be more efficient.

Definition 7. *$W^0(C|G)$ is the weight of C on the compatible underlying graph G under $\ell = 0$. In other words, $W^0(C|G) = \prod_{i=1}^{L-1} d(v_i)$ if $C = (v_1, \dots, v_L)$ and $d(v)$ is the out-degree of $v \in G$.*

We use the same notations as in Lemma 1 to state the result of this section:

Lemma 3. *If one sets*

$$P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big) = \min\left(1, \frac{q(S_{\mathrm{n}})W(C_{\mathrm{n}}|G_{\mathrm{n}})W^0(C_{\mathrm{n}}|G_{\mathrm{n}})^{-1}}{q(S_{\mathrm{o}})W(C_{\mathrm{o}}|G_{\mathrm{o}})W^0(C_{\mathrm{o}}|G_{\mathrm{o}})^{-1}}\right), \tag{7}$$

then q is the stationary distribution of the extended RG* algorithm.

Proof. We show that q satisfies the same "locale" balance equations (6). The probability of acceptance P_{acc} is given, so we have to compute the new probabilities of generating our objects under the extended RG* algorithm, which correspond to formulas (2), (3) and (4).

Since only the generation of the underlying graphs changes in this algorithm, formula (4) still holds with the same definition for the w_i. Note that w_i is not bounded by k anymore, but rather by the out-degree $d(v_i)$ of v_i. More generally, we note $d(v)$ the out-degree of v, with no possible confusion on the underlying graph that is considered.

We need some notations: let C be a polymer, G a compatible underlying graph, and $1 \leq i \leq Q$. We note $\alpha_i = \binom{Q}{i}^{-1}$, $\beta_i = \binom{Q-1}{i-1}^{-1}$. $\widetilde{C}$ is C minus its extremity that is not the root of G, $k_i(G) = |\{v \in G : d(v) = i\}|$, $q_i(G|C) = |\{v \in G \backslash \widetilde{C} : d(v) = i\}|$ and $\tilde{q}_i(G|C) = |\{v \in \widetilde{C} : d(v) = i\}|$.

Now if we turn our attention to the generation of a general underlying graph, we see that formula (2) becomes

$$P_{\mathrm{u}}(G) = \frac{1}{\gamma} \prod_{v \in G} p_{d(v)} \alpha_{d(v)} = \frac{1}{\gamma} \prod_{i=1}^{Q} (p_i \alpha_i)^{k_i(G)}.$$

Indeed, we assign the out-degree d to the vertex v with probability p_d, and then, there are $1/\alpha_d$ different ways to choose its neighbors. Note that $p_i = 0$ implies $k_i(G) = 0$, and we adopt the convention that $0^0 = 1$. For the same reasons, formula (3) becomes

$$P_{\mathrm{c}}(G|C) = \frac{1}{2} \prod_{v \in G \backslash \widetilde{C}} p_{d(v)} \alpha_{d(v)} \prod_{v \in \widetilde{C}} p_{d(v)} \beta_{d(v)} = \frac{1}{2} \prod_{i=1}^{Q} \left[(p_i \alpha_i)^{q_i(G|C)} (p_i \beta_i)^{\tilde{q}_i(G|C)} \right].$$

Using $|\widetilde{C}| = L - 1$, $k_i(G) = q_i(G|C) + \widetilde{q}_i(G|C)$, $\beta_i = Q\alpha_i/i$ and $W^0(G|C) = \prod_{v \in \widetilde{C}} d(v) = \prod_1^Q i^{\tilde{q}_i(G|C)}$, we finally can rewrite

$$P_{\mathrm{c}}(G|C) = \frac{\gamma Q^{L-1}}{2} P_{\mathrm{u}}(G) W^0(C|G)^{-1}.$$

Putting all these formulas together, it is then easy to check that (6) amounts to

$$\frac{P_{\mathrm{acc}}\big((S_{\mathrm{o}}, G_{\mathrm{o}}), (S_{\mathrm{n}}, G_{\mathrm{n}})\big)}{P_{\mathrm{acc}}\big((S_{\mathrm{n}}, G_{\mathrm{n}}), (S_{\mathrm{o}}, G_{\mathrm{o}})\big)} = \frac{q(S_{\mathrm{n}}) W(C_{\mathrm{n}}|G_{\mathrm{n}}) W^0(C_{\mathrm{n}}|G_{\mathrm{n}})^{-1}}{q(S_{\mathrm{o}}) W(C_{\mathrm{o}}|G_{\mathrm{o}}) W^0(C_{\mathrm{o}}|G_{\mathrm{o}})^{-1}}.$$

The acceptance probability chosen exactly satisfies this equality. □

Remark: When p is supported by only one point k, the terms W^0 cancel out, and we find the probability of acceptance for the standard RG* algorithm.

5. Implementation

We have mentioned at several occasions throughout this paper that the original algorithm in [3] is actually an efficient implementation of the RG* algorithm as defined in the present paper. In this section, we present this implementation, cast in our framework.

The inefficiency of the RG* algorithm comes from the generations of the underlying graphs. As we have defined them, the only way they interact with the rest of the system is through their roots. The root of an underlying graph has to be a free vertex, but once it is chosen, we do not care whether vertices which we add belong to existing polymers or not, nor do we care whether they are at a distance larger than L from the root. However, we will never visit such vertices in the process of growing a polymer.

We have already stressed out that the underlying graph and the growth of the polymer are independent concepts. This has made possible to define and study them separately. But since they are independent, nothing forbids to generate the underlying graph while growing the polymer. This way, we are sure to only generate

the parts of the underlying graph that we visit during the growth procedure. So the idea is just to grow the polymer as usual, and to complete the underlying graph when the current endpoint of the polymer has not been assigned out-edges yet.

If $d(v)$ is the out-degree of vertex v, the entangled RG* algorithm becomes as follows:

---

1. Initialization:
   (a) Pick an unoccupied vertex r uniformly at random
   (b) Set $T = \{r\}$ and $V = E = \emptyset$
   (c) Set $\overline{C} = (r), D_1 = \emptyset$ and $L_{\max} = 1$
2. At each step, with $\overline{C} = (v_1, \dots, v_i)$:
   (a) If $i = L$, stop, $\overline{C}$ is a complete polymer
   (b) If v_i has not been assigned out-edges, i.e. $v_i \in T$:
      i. Set $d(v_i) = \kappa$ with probability π_κ
      ii. Choose uniformly at random $d(v_i)$ different vertices among the Q neighbors of v_i in $\mathcal{G}$
      iii. For each vertex v chosen:
         (A) Add the edge $v_i \to v$ to E
         (B) If $v \notin T$ and $v \notin E$, add v to T
      iv. Remove v_i from T and add it to V
   (c) If $|D_i| = d(v_i)$, recoil:
      i. If $i > \max(1, L_{\max} - \ell)$, set $\overline{C} = (v_1, \dots, v_{i-1})$ and go to 2
      ii. Otherwise, stop, the generation has failed
   (d) Else keep picking uniformly at random a vertex v neighbor of v_i in G and not in D_i, add it to D_i, and stop when either v is unoccupied or $|D_i| = d(v_i)$:
      i. If v is unoccupied
         (A) Set $\overline{C} = (v_1, \dots, v_i, v), D_{i+1} = \emptyset$ and update $L_{\max}$
         (B) If $v \notin T$ and $v \notin E$, add v to T
         (C) Go to 2
      ii. Else, $|D_i| = d(v_i)$, and recoil as specified in 2.(c)

At the end of this procedure, it can happen that the underlying graph is not complete: some vertices can have not been assigned out-edges. It is then important to keep the current partial underlying graph in memory, because this graph will be further completed if the next step of the algorithm is to compute the weights. Indeed, the same idea applies in this case: when we compute the weights, we do as usual, and we complete the current partial underlying graph when required.

Acknowledgment. The author would like to thank Persi Diaconis and Philippe Robert for their persistent support and interest in this work, as well as Frédéric Meunier for very fruitful discussions.

References

1. Julian Besag and Peter J. Green, *Spatial statistics and bayesian computation*, Journal of the Royal Statistical Society **55** (1993), no. 1, 25–37.
2. S. Consta, T.J.H. Vlugt, J.W. Hoeth, B. Smit, and D. Frenkel, *Recoil growth algorithm for chain molecules with continuous interaction*, Molecular Physics **97** (1999), no. 12, 1243–1254.

3. S. Consta, N.B. Wilding, D. Frenkel, and Z. Alexandrowicz, *Recoil growth: an efficient simulation method for multi-polymer systems*, Journal of Chemical Physics **110** (1999), no. 6, 3320–3228.
4. Persi Diaconis and Gilles Lebeau, *Metropolis*, Available at http://www-stat.stanford.edu/∼cgates/PERSI/year.html (2007).
5. R.G. Edwards and A. D. Sokal, *Generalization of the fortuin-kasteleyn-swendsen-wang representation and monte carlo algorithm*, Physical Review Letters **38** (1988), 2009–2012.
6. Ole J. Heilmann and Elliott H. Lieb, *Theory of monomer-dimer systems*, Communication in Mathematical Physics **25** (1972), no. 3, 190–232.
7. David M. Higdon, *Auxiliary variable methods for markov chain monte carlo with applications*, Journal of the American Statistical Association **93** (1998), no. 442, 585–595.
8. Neal Madras and Gordon Slade, *The self-avoiding walk*, Birkhauser, 1993.
9. Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller, *Equations of state calculations by fast computing machines*, Journal of Chemical Physics **21** (1953), no. 6, 1087–1092.
10. J.I. Siepmann, *A method for the direct calculation of chemical potentials for dense chain systems*, Molecular Physics **70** (1990), 1145–1158.
11. J.I. Siepmann and D. Frenkel, *Configurational-bias monte carlo - a new sampling scheme for flexible chains*, Molecular Physics **75** (1992), 59–70.
12. F. Simatos, *A variant of the Recoil-Growth algorithm to generate multi-polymer systems*, Fifth Colloquium on Mathematics and Computer Science, 2008.
13. Alan D. Sokal, *Monte carlo methods for the self-avoiding walk*, Nuclear Physics B - Proceedings Supplements **47** (1996), no. 1-3, 172–179.
14. J. van den Berg and R. Brouwer, *Random sampling for the monomer-dimer model on a lattice*, Tech. Report PNA-R9917, CWI, 1999.

E-mail address: `florian.simatos@inria.fr`
URL: `http://www-rocq.inria.fr/∼simatos`

INRIA, domaine de Voluceau, B.P. 105, 78153 Le Chesnay Cedex, France